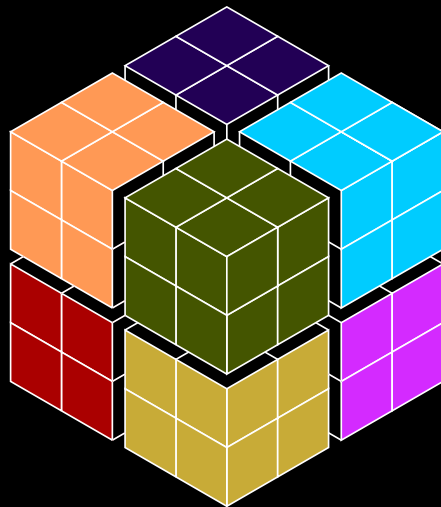


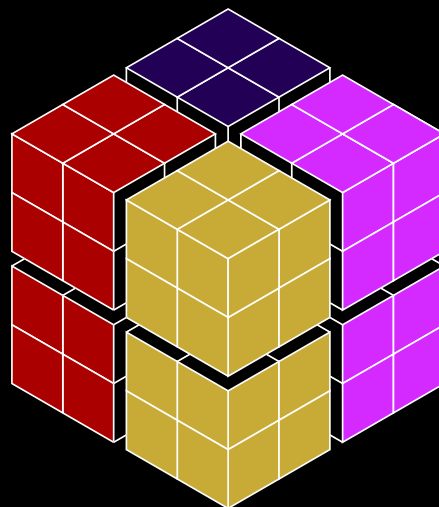
“I have had my results for a long time, but I do not yet know how I am to arrive at them.”

–Carl Friedrich Gauss, 1777-1855

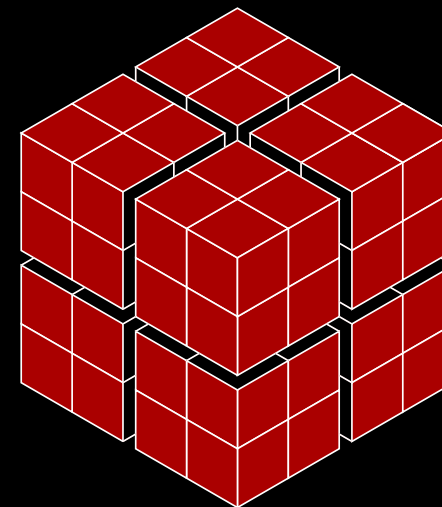
## Block-Based Analysis of Scientific Data



8 processes



4 processes



1 process

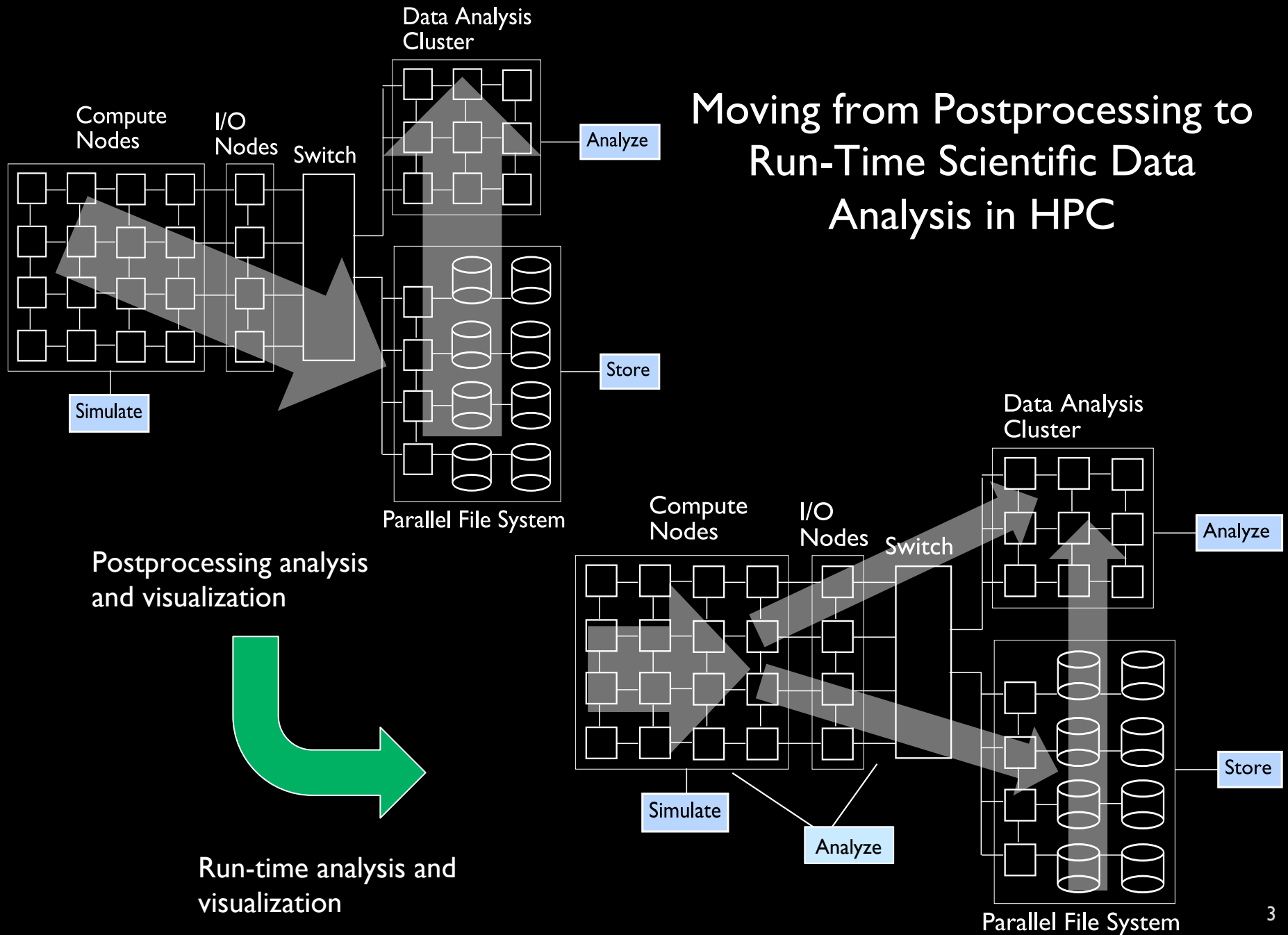
*Parallel data analysis consists of decomposing a problem into **blocks**, operating on them, and communicating between them.*

Tom Peterka

[tpeterka@mcs.anl.gov](mailto:tpeterka@mcs.anl.gov)

# Preliminaries

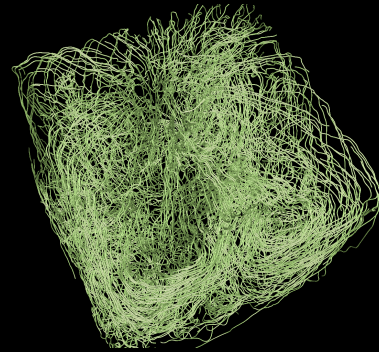
# Moving from Postprocessing to Run-Time Scientific Data Analysis in HPC



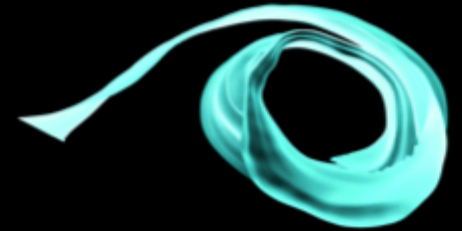
# Parallel Data Analysis

- Big science => big data, big machines
- Most analysis algorithms are not up to speed
  - Either serial, or
  - Overheads kill scalability
- Solutions
  - Process data closer to the source
  - Write scalable analysis algorithms
  - Parallelize in various forms

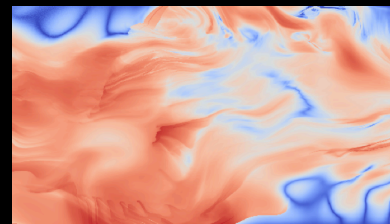
*Question: What is the best abstraction to express parallelism?*



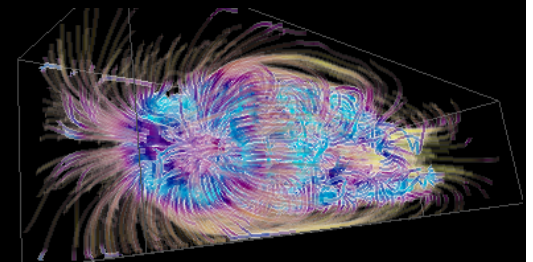
Streamlines and pathlines



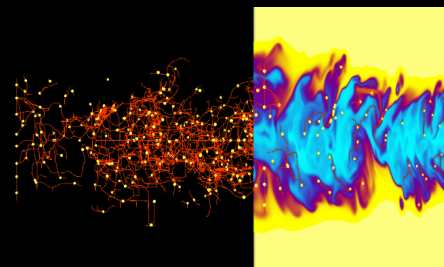
Stream surfaces



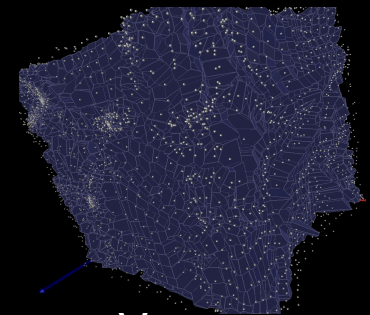
FTLE



Information entropy



Morse-Smale complex



Voronoi Tessellation

# Abstractions Matter: Think Blocks, not Tasks

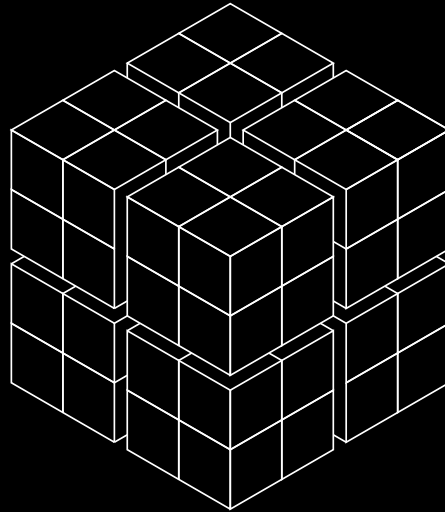
- Block = unit of decomposition
- Block size, shape can be configured
  - From coarse to fine
  - Regular, adaptive, KD-tree
- Block placement is flexible, dynamic
  - Blocks per task
  - Tasks per block
  - Memory / storage hierarchy
- Data is first-class citizen
  - Separate operations per block
  - Thread safety

*Parallel data analysis consists of decomposing a problem into blocks, operating on them, and communicating between them.*

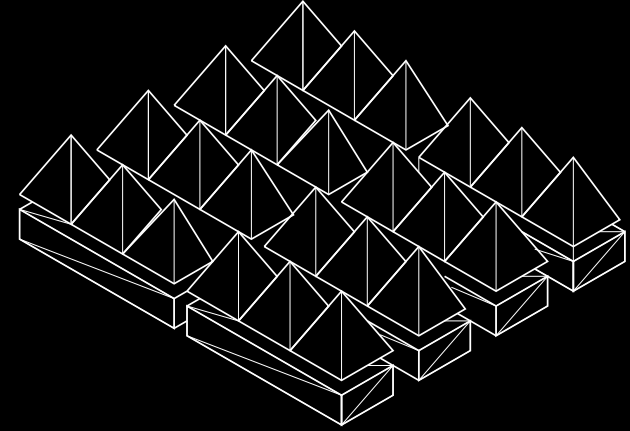
# The What and Why of a Block-Based Approach

# Partition Data Into Blocks

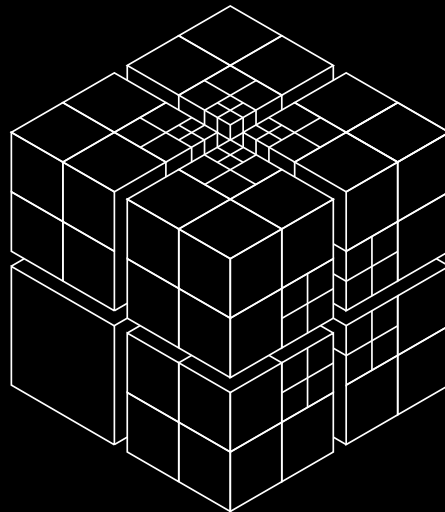
The block is the basic unit of data decomposition. Original dataset is decomposed into generic subsets called blocks, and associated analysis items live in the same blocks. Blocks don't have to be "blocky." Any subdivision of data (eg., a set of graph nodes, a group of particles, etc.) is a block.



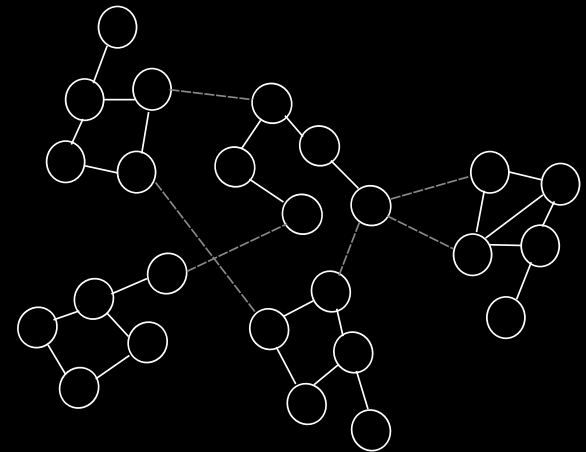
Structured Grid



Unstructured Mesh



AMR Grid



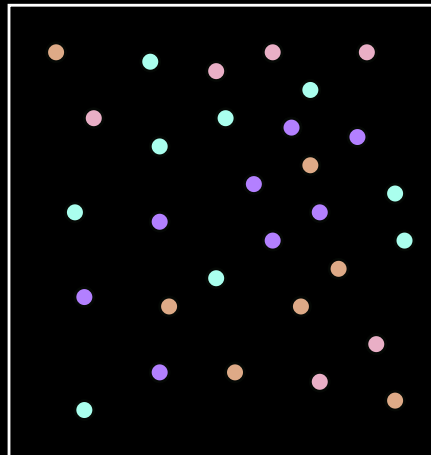
Graph

# Create Multiple Decompositions

Uses:

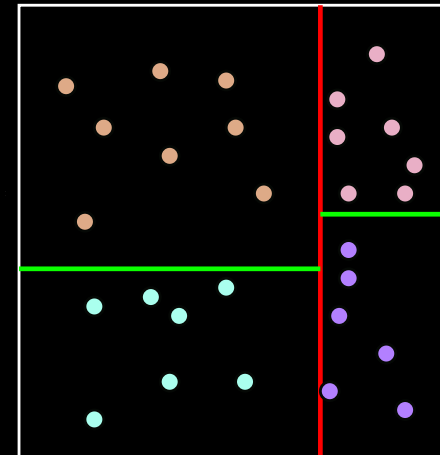
1. Organize input (upper right)
2. Second decomposition suited for particular analysis (lower right)
3. Comparing multiple unrelated data domains (not shown)

Original data  
Arbitrary decomposition



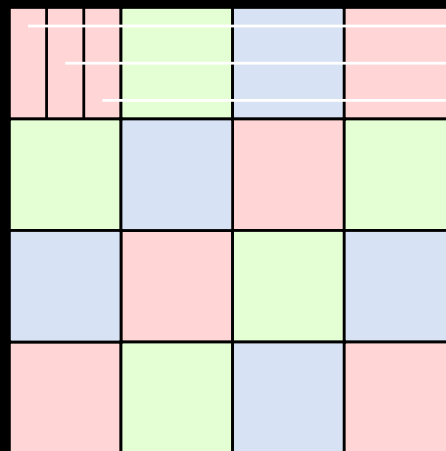
4 blocks indicated by color.  
No spatial locality assumed.

Kd-tree decomposition



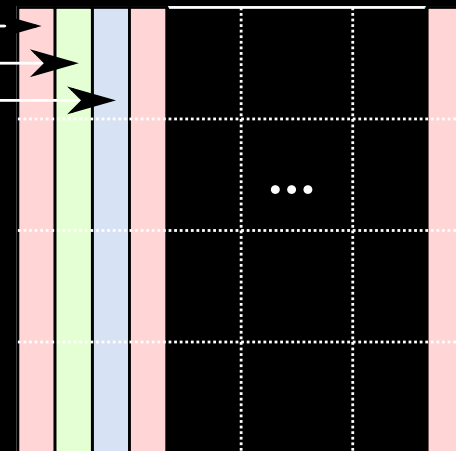
4 new blocks spatially contiguous  
and load balanced by number of  
objects in each.

Original block decomposition



16 blocks, 3 procs indicated by color

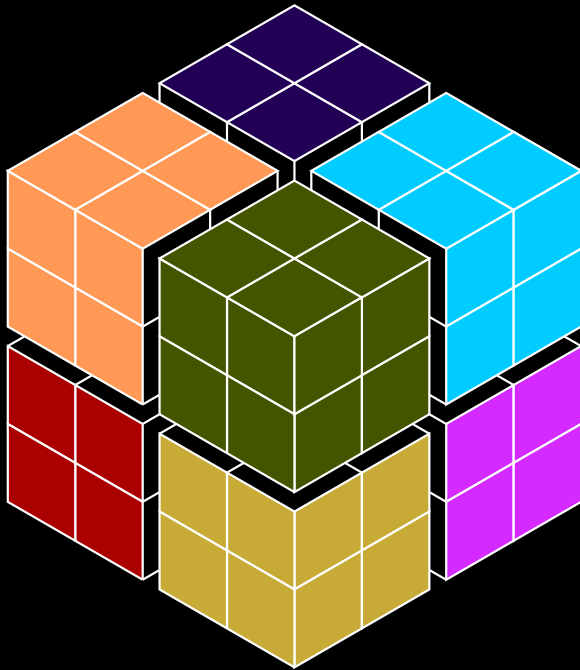
Slab or pencil decomposition for FFT



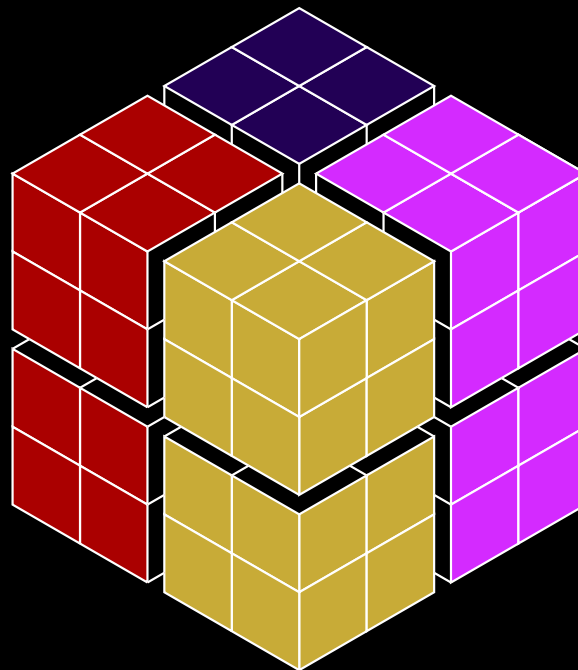
Need not be same number of blocks



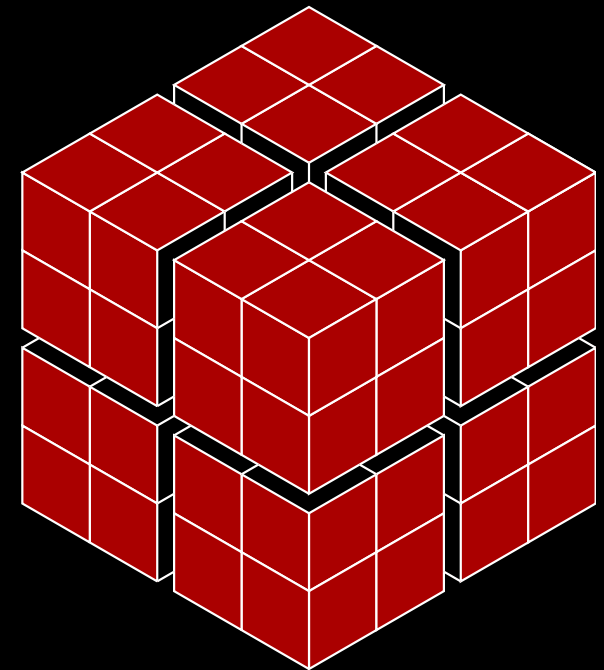
## Distinguish Between Blocks and Processes



8 processes



4 processes

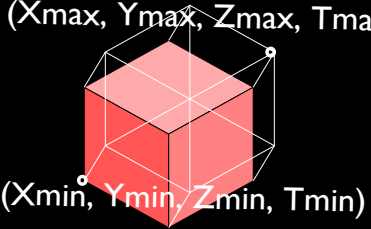
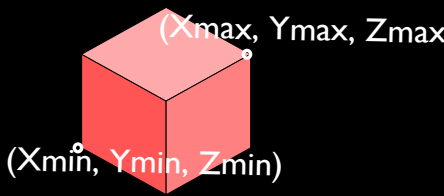
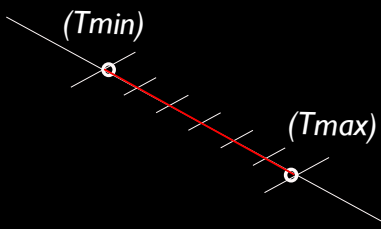
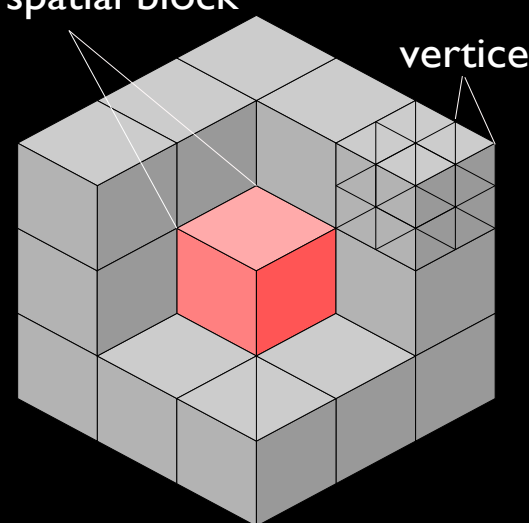
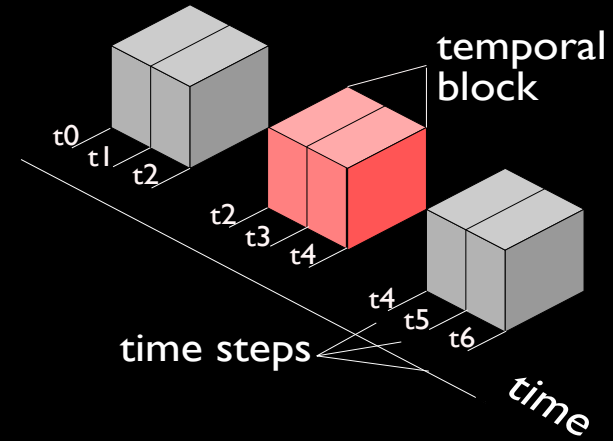


1 process

All data movement operations are per **block**; blocks exchange information with each other using regular communication patterns. Runtime manages and optimizes exchange between processes based on the process assignment. This allows for flexible process assignment as well as easy debugging.

# Handle Time

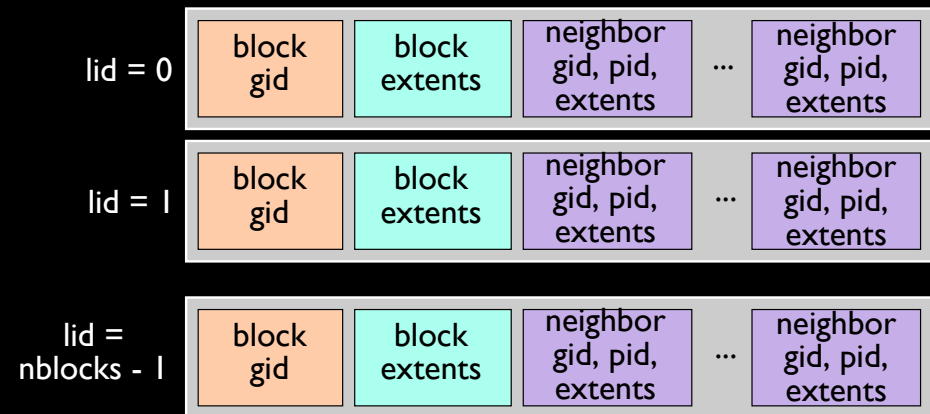
- Time often goes forward only
- Usually do not need all time steps at once

4D	=	3D	×	1D
<p>(Xmax, Ymax, Zmax, Tmax)</p>  <p>(Xmin, Ymin, Zmin, Tmin)</p> <p>4D Block</p>		<p>(Xmax, Ymax, Zmax)</p>  <p>(Xmin, Ymin, Zmin)</p> <p>3D Spatial Extent</p>		<p>(Tmin)</p>  <p>(Tmax)</p> <p>1D Temporal Extent</p>
<p>4D Neighborhood (not drawn)</p>		<p>spatial block</p>  <p>vertices</p> <p>3D Spatial Neighborhood</p>		 <p>temporal block</p> <p>time steps</p> <p>time</p> <p>1D Temporal Neighborhood</p>

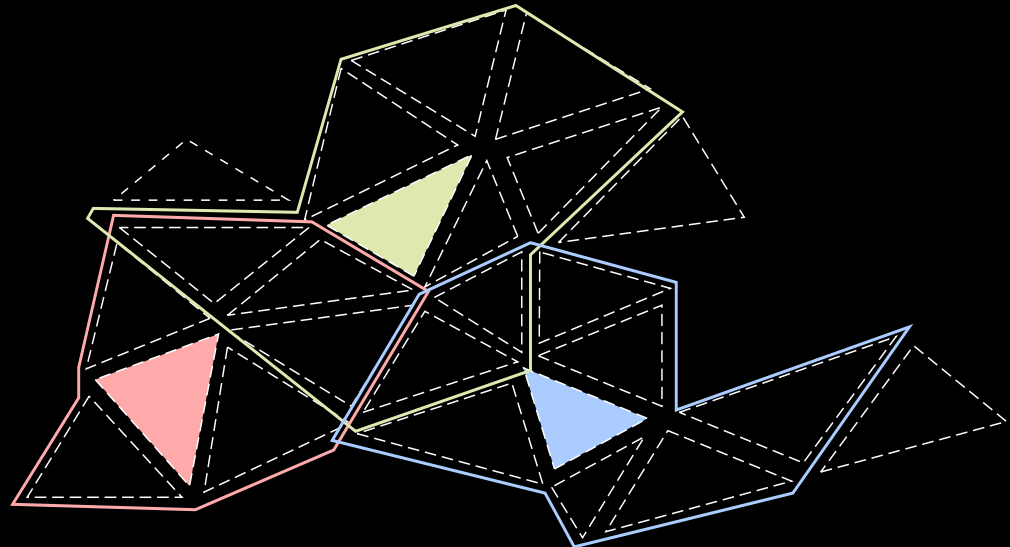
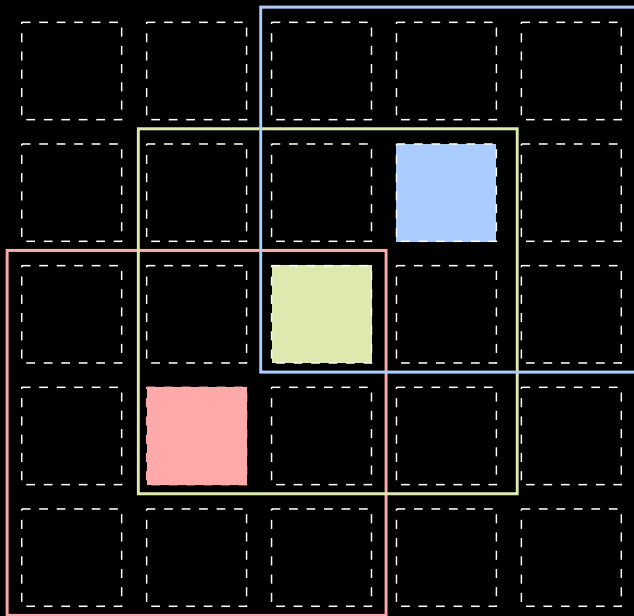
Hybrid 3D/4D time-space decomposition. Time-space is represented by 4D blocks that can also be decomposed such that time blocking is handled separately.

# Group Blocks into Neighborhoods

- Limited-range communication
- Allow arbitrary groupings
- Distributed, local data structure and knowledge of other blocks (not master-slave global knowledge)

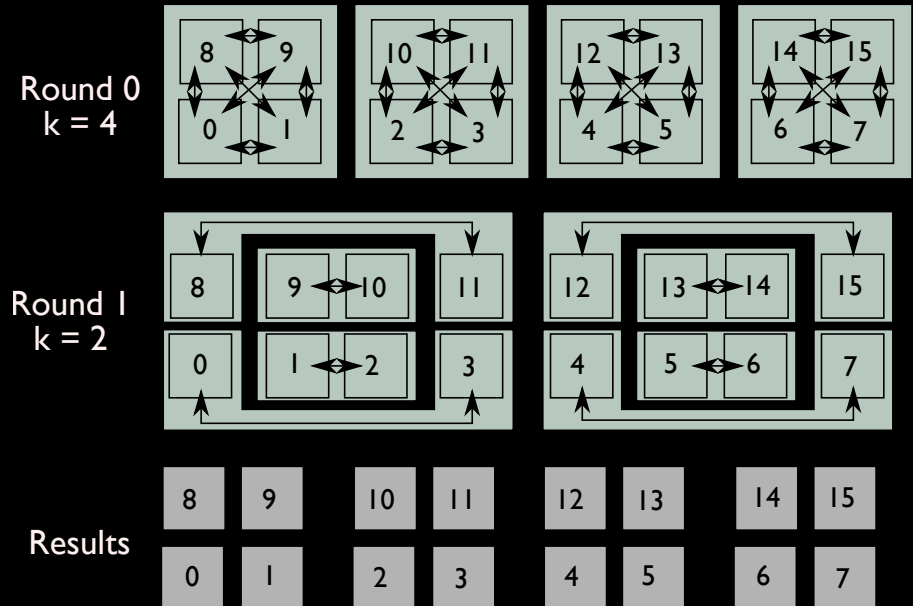
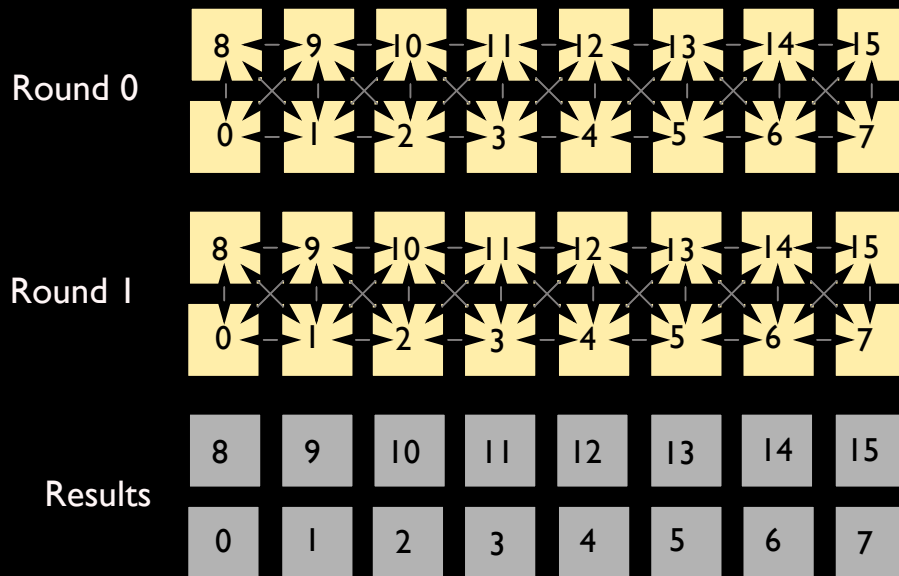


gid = global block identification  
lid = local block identification  
pid = process identification

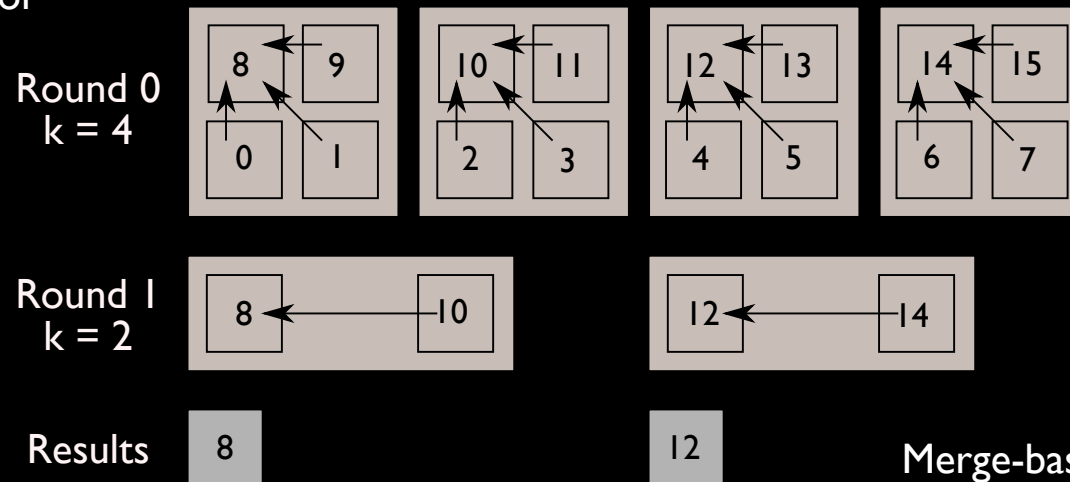


Two examples of 3 out of a total of 25 neighborhoods

# Communicate Locally and Globally Between Blocks



Nearest neighbor



Swap-based reduction

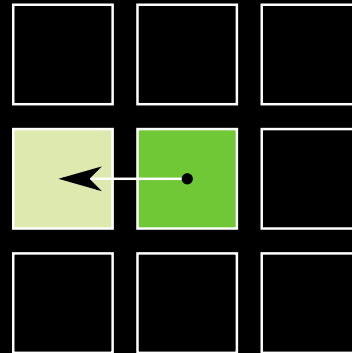
Merge-based reduction

# Different Neighborhood Communication Patterns

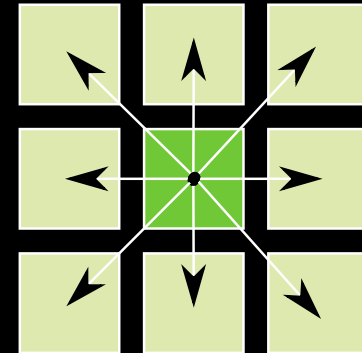
Provide point to point and different varieties of collectives within a neighborhood by enqueueing and subsequently exchanging items (2 steps).

## How to enqueue items for neighbor exchange

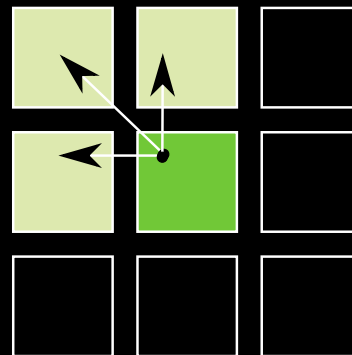
- Send to a particular neighbor or neighbors, send to all nearby neighbors, send to all neighbors
- Support for periodic boundary conditions involves tagging which neighbors are periodic and calling user-defined transform on objects being sent to them



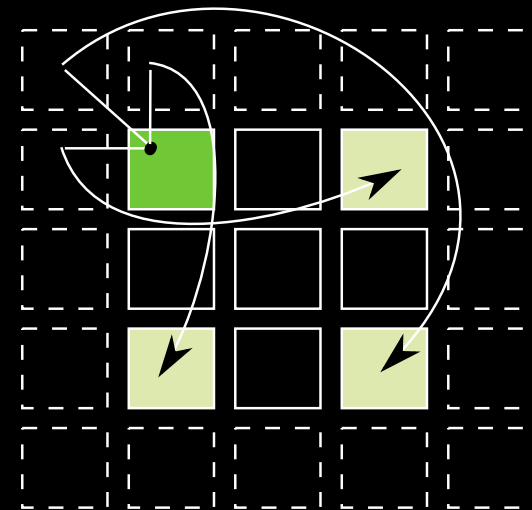
Send to only specific neighbors, indicated in various ways



Send to all neighbors



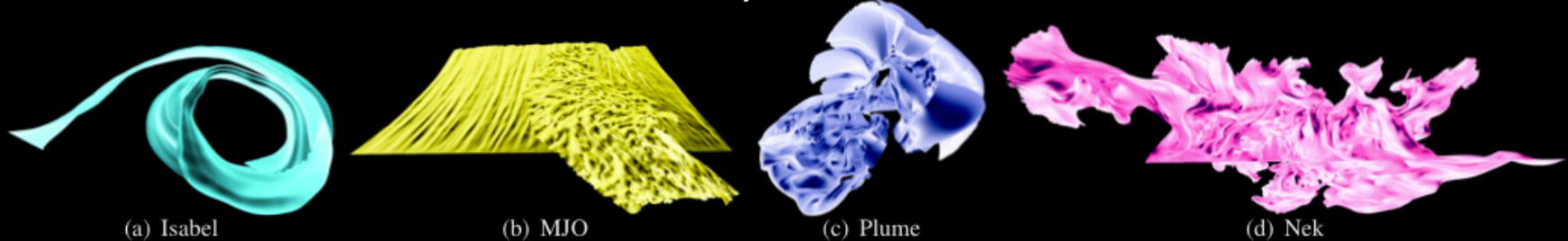
Send to all neighbors near enough to a target point



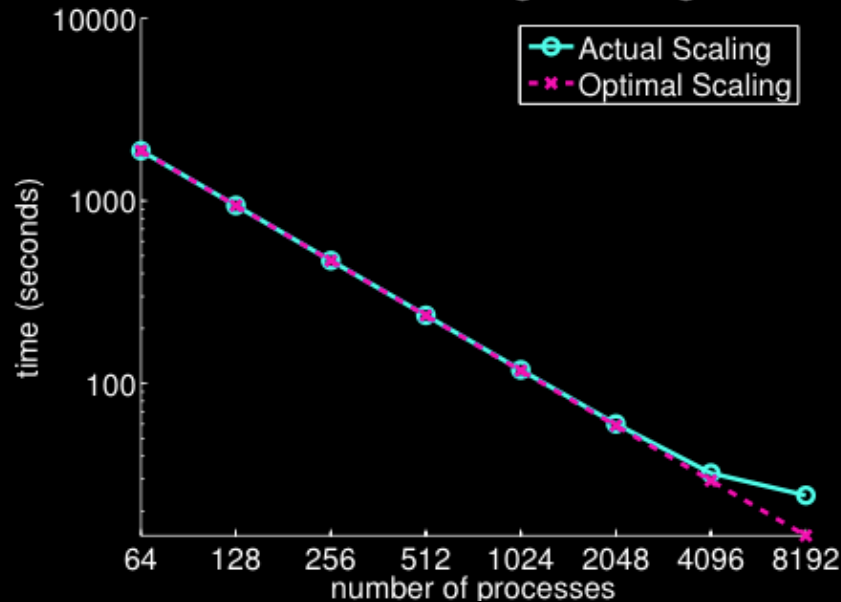
Support for wraparound neighbors (periodic boundary conditions)

# Migrate Blocks for Load Balancing (for Stream Surfaces)

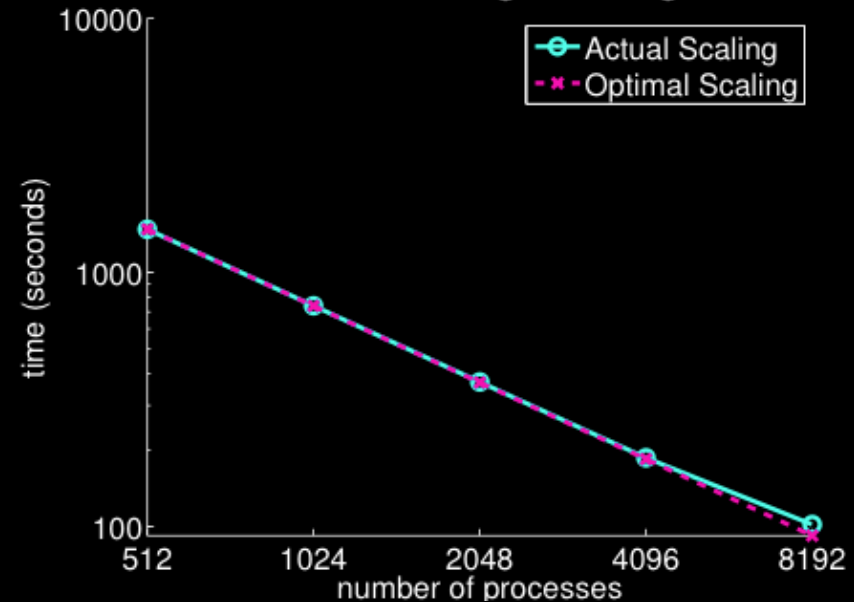
Courtesy Kewei Lu



*Plume: Strong Scaling*



*Nek: Strong Scaling*

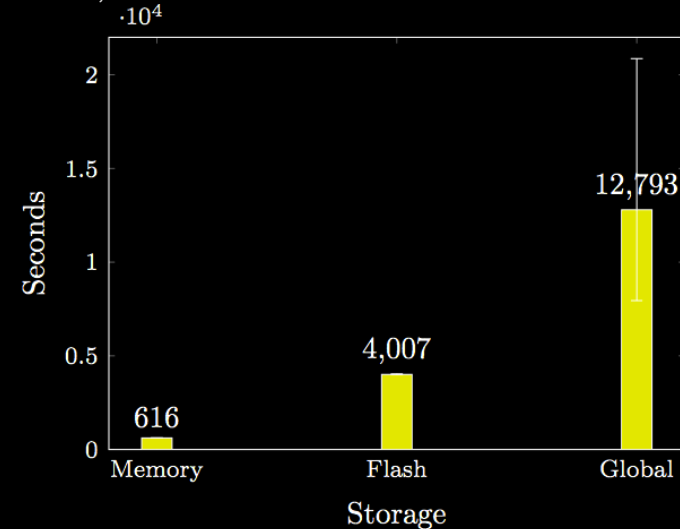
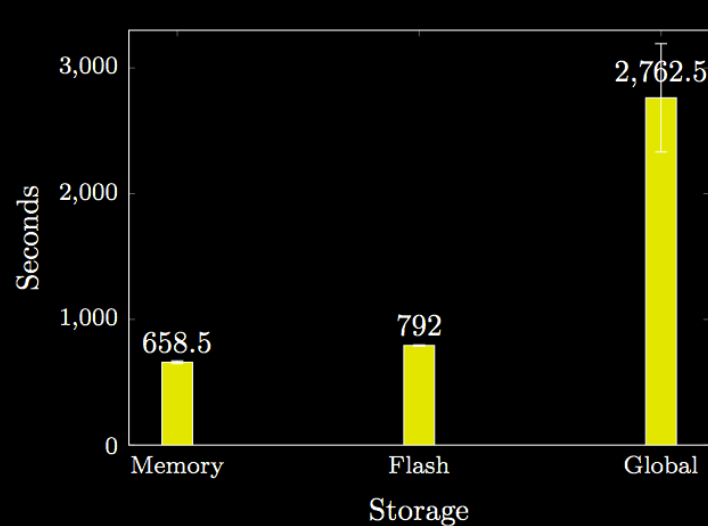


Left: 64 surfaces each seeded with 512 particles are advected in a  $504 \times 504 \times 2048$  simulation of a solar flare. Right: 64 surfaces each with 2K seeds in a  $2K \times 2K \times 2K$  Nek5000 thermal hydraulics simulation. Time excludes I/O.

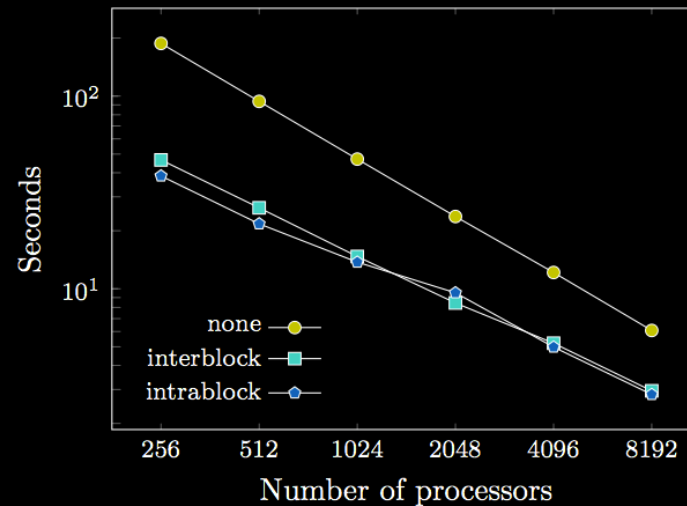
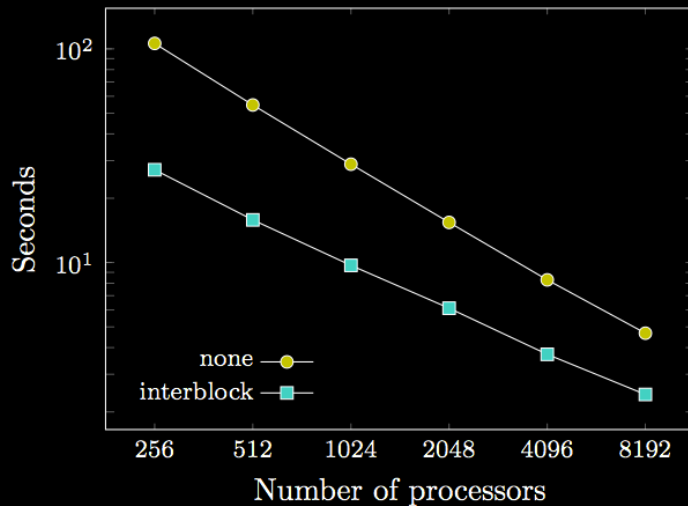
Lu et al., Scalable Computation of Stream Surfaces on Large Scale Vector Fields, SC14.

# Easily Write OOC and Multithreaded Algorithms

With Dmitriy Morozov, LBNL



Left: In- and out-of-core performance of watershed segmentation. Right: In- and out-of core performance of Voronoi tessellation..



Left: Automatic threading of Voronoi tessellation. Right: comparison between manual and automatic threading of density estimation.

One Example in Greater Detail

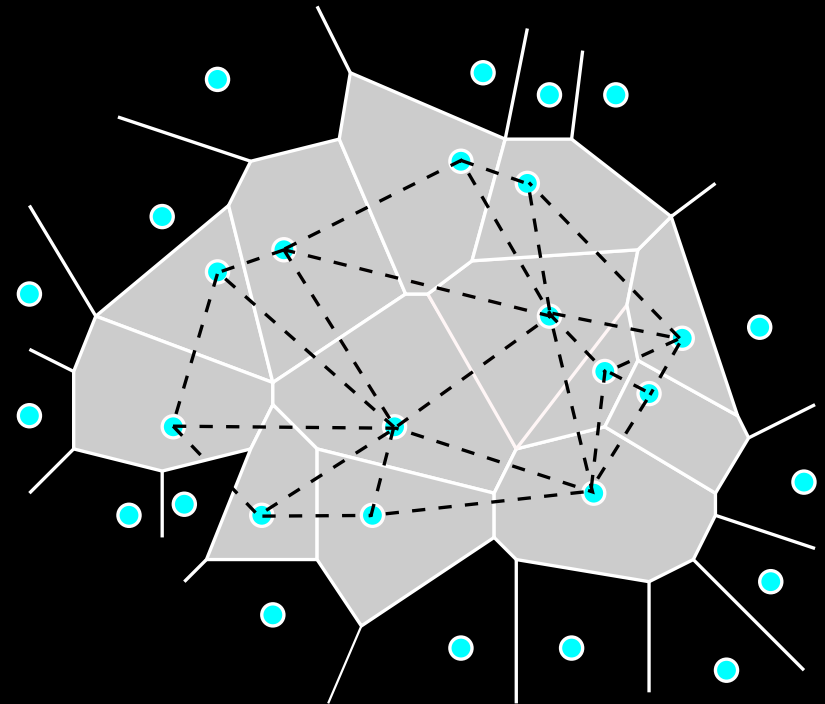


# Parallel Tessellation

We developed a prototype library for computing in situ Voronoi and Delaunay tessellations from particle data and applied it to cosmology, molecular dynamics, and plasma fusion.

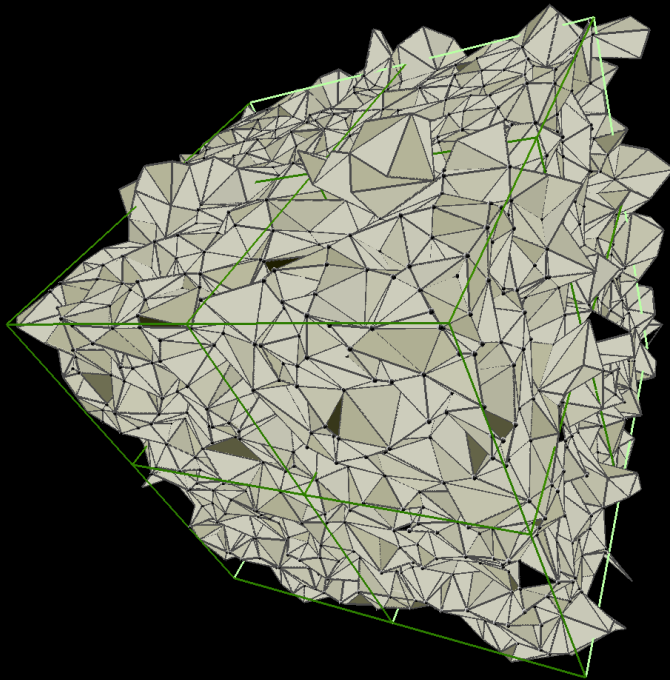
## Key Ideas

- Mesh tessellations convert sparse point data into continuous dense field data.
- Meshing output of simulations is data-intensive and requires supercomputing resources
- No large-scale data-parallel tessellation tools exist.
- We developed such a library, tess.
- We achieved good parallel performance and scalability.
- Widespread GIS applicability in addition to the datasets we tested.

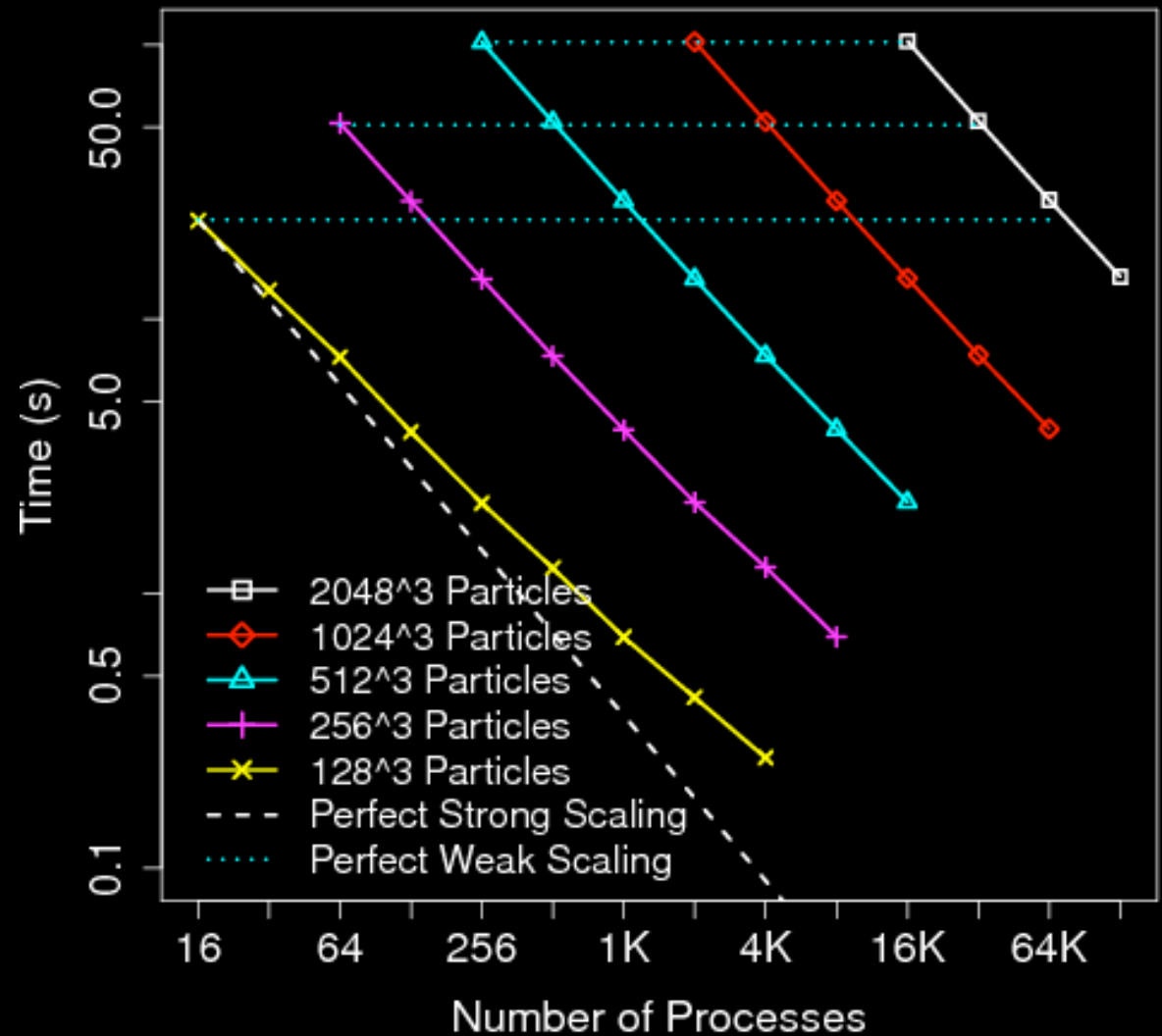


# Scalability

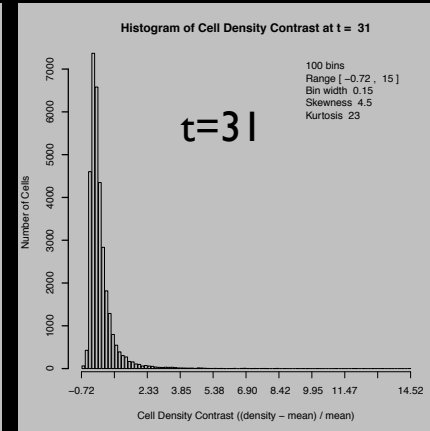
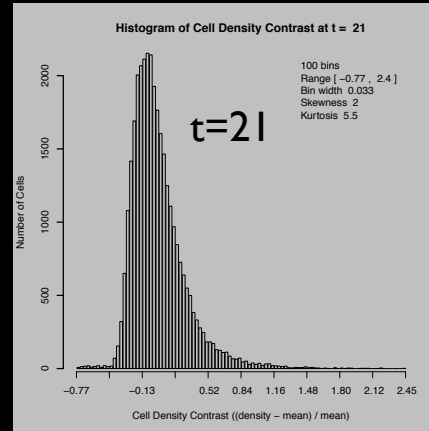
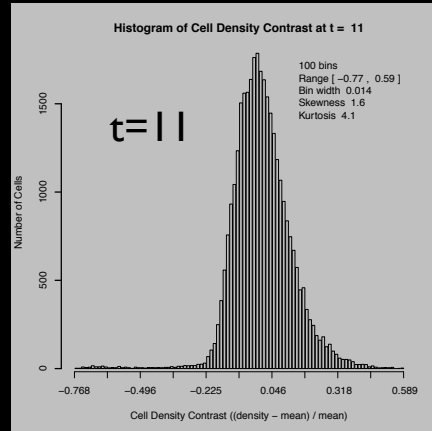
## Strong and Weak Scaling with CGAL



Strong and weak scaling for up to  $2048^3$  synthetic particles and up to 128K processes (excluding I/O) shows up to 90% strong scaling and up to 98% weak scaling.

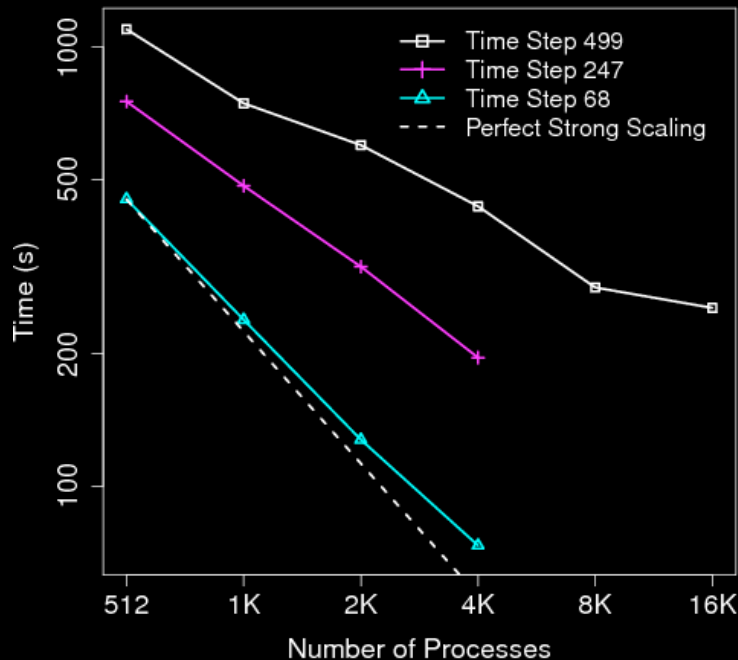


# Applications in Cosmology

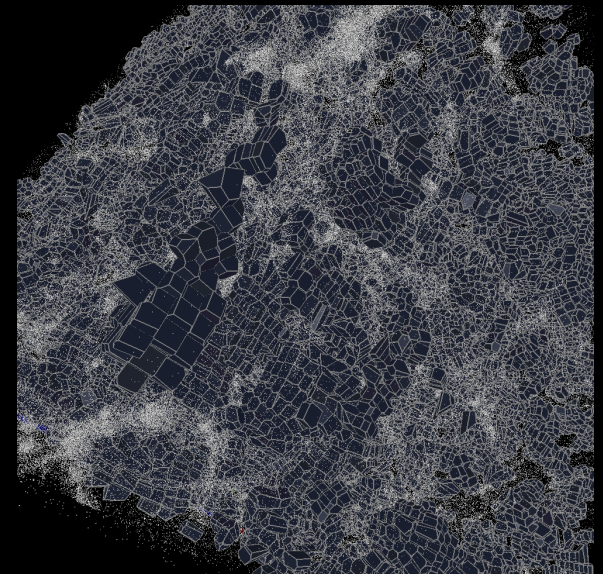


Temporal structure dynamics: As time progresses, the range of cell volume and density expands, kurtosis and skewness increases, consistent with the governing physics.

Strong Scaling of HACC Data on Mira



Strong scaling (excluding I/O time) using CGAL for three time steps of HACC data of  $1024^3$  particles. At later time steps, particles cluster into extremely dense and sparse regions, affecting load balance and reducing efficiency from 77% at  $t=68$  to 14% at  $t = 499$ .



# Summary

# Recap

Block abstraction for parallelizing data analysis allows one to:

- Decompose data into blocks
- Assign blocks to processing elements
- Have several decompositions at once
- Overload blocks, migrate blocks between processing elements
- Communicate between blocks
- Migrate blocks in and out of core
- Thread blocks with finer-grained processing elements

All made possible by choosing blocks as the parallel abstraction

*Think Blocks!*

# Further Reading

## DIY

- Peterka, T., Ross, R., Kendall, W., Gyulassy, A., Pascucci, V., Shen, H.-W., Lee, T.-Y., Chaudhuri, A.: Scalable Parallel Building Blocks for Custom Data Analysis. Proceedings of Large Data Analysis and Visualization Symposium (LDAV'11), IEEE Visualization Conference, Providence RI, 2011.
- Peterka, T., Ross, R.: Versatile Communication Algorithms for Data Analysis. 2012 EuroMPI Special Session on Improving MPI User and Developer Interaction IMUDI'12, Vienna, AT.

## DIY applications

- Peterka, T., Kwan, J., Pope, A., Finkel, H., Heitmann, K., Habib, S., Wang, J., Zagaris, G.: Meshing the Universe: Integrating Analysis in Cosmological Simulations. Proceedings of the SCI2 Ultrascale Visualization Workshop, Salt Lake City, UT.
- Chaudhuri, A., Lee-T.-Y., Zhou, B., Wang, C., Xu, T., Shen, H.-W., Peterka, T., Chiang, Y.-J.: Scalable Computation of Distributions from Large Scale Data Sets. Proceedings of 2012 Symposium on Large Data Analysis and Visualization, LDAV'12, Seattle, WA.
- Peterka, T., Morozov, D., Phillips, C.: High-Performance Computation of Distributed-Memory Parallel 3D Voronoi and Delaunay Tessellation. Proceedings of SCI4, New Orleans, LA, 2014.
- Lu, K., Shen, H.-W., Peterka, T.: Scalable Computation of Stream Surfaces on Large Scale Vector Fields. Proceedings of SCI4, New Orleans, LA, 2014.

“The purpose of computing is insight, not numbers.”

–Richard Hamming, 1962

## Acknowledgments:

### Facilities

Argonne Leadership Computing Facility (ALCF)  
Oak Ridge National Center for Computational Sciences (NCCS)

### Funding

DOE SDMAV Exascale Initiative  
DOE Exascale Codesign Center  
DOE SciDAC SDAV Institute  
<https://bitbucket.org/diatomic/diy>

Tom Peterka

[tpeterka@mcs.anl.gov](mailto:tpeterka@mcs.anl.gov)